Memory Allocation

Memory Allocation — Dynamic

= allocate + free memory chunks of arbitrary size at arbitrary points in time
  - almost every program uses it (heap)
  - don't have to statically specify complex data structures
  - can have data grow as function of input size
  - kernel itself uses dynamic memory allocation for its data structures
- **Implementation**: has huge impact on performance, both in user and kernel space
- **Fact**: it is impossible to construct memory allocator that always performs well

→ need to understand trade-offs to pick good allocation strategy

Dynamic Memory Allocation — Principle

- **Initial**: pool of free memory
- **Tasks**:
  - satisfy arbitrary `allocate` + `free` requests from pool
  - track which parts are in use/are free
- **Restrictions**:
  - cannot control order/number of requests
  - cannot move allocated regions → fragmentation = core problem!

Dynamic Memory Allocation — Bitmap

- **Idea**:
  - divide memory in allocation units of fixed size
  - use bitmap to keep track if allocated (1) or free (0)
- **Problem**: needs additional data structure to store allocation length (otherwise cannot infer whether two adjacent allocations belong together or not from bitmap)

Dynamic Memory Allocation — List

- **Method 1**: use one list-node for each allocated data
  - *extra space* needed for list
  - allocation lengths already stored
- **Method 2**: use one list-node for each unallocated data
  - can keep list in unallocated area (store size of free area + pointer to next free area in free area)
  - *additional data structure* needed to store allocation lengths
  - can search for free space with low overhead
- **Method 3**: both

Dynamic Memory Allocation — Problems

- **Fragmentation** is hard to handle
- **Factors** needed for fragmentation to occur:
  - *different lifetimes*
  - *different sizes*
  - *inability to relocate previous allocations*
- all fragmentation factors present in dynamic memory allocators!

Allocation — Best fit vs. Worst fit

- **Idea**: keep large free memory chunks together for larger allocation requests that may arrive later
- **Best-fit**: allocate smallest free block large enough to store allocation request
  - must search entire list
  - *problem*: sawdust — remainder so small that over time left with unusable sawdust everywhere
  - *idea*: minimize sawdust by turning strategy around
- **Worst-fit**: allocate largest free block
  - must search entire list
  - *reality*: worse fragmentation than best-fit

Allocation — First fit

- **Idea**: if fragmentation occurs with best and worst fit, optimize for allocation speed
- **Principle**: allocate first hole big enough
  - fastest allocation policy
  - produced leftover holes of variable size
  - *reality*: almost as good as best-fit

First Fit — Variants

- **first-fit sorted by address order**
- **LIFO first-fit**
- **next fit**

Allocation — Buddy allocator

- **Idea**: allocate memory in powers of 2
  - all chunks have fixed 2^n-size → allocation request rounded up to next-higher power of 2
  - all chunks naturally aligned
- **no sufficiently small block available**:
  - select larger available chunk, split into two same-sized buddies
  - continue until appropriately sized chunk is available
- **two buddies both free** (2^n): merge to 2^{n+1}-chunk

Real Program Patterns

- **Ramps**: accumulate data monotonically over time
- **Peaks**: allocate many objects, use briefly, then free all
- **Plateaus**: allocate many objects, use for long time

Allocation — Slabs

- kernel often allocates/frees memory for few, specific data objects of fixed size
- **Slab**: multiple pages of contiguous physical memory
  - linux: uses buddy allocator as underlying allocator for slabs
- **Cache**: one or multiple slabs
  - stores only one kind of object (fixed size)

Summary

- dynamic memory means allocating and freeing memory chunks of different sizes at any time
- impossible to construct memory allocator that always performs well
- typical dynamic memory data structures:
  - bitmaps
  - lists
- simple, well-performing allocation strategies:
  - best-fit
  - first-fit
- advanced strategies:
  - buddy-allocator
  - slab-allocator